UC Thesis

Improving Prescripted Agent Behavior with Neuroevolution

Undergraduate Thesis - Ryan Cornelius

Faculty Advisor - Risto Miikkulainen

Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712-0233

Abstract

Machine learning can increase the appeal of video games by allowing agents to adapt in response to the player. Therefore, methods need to be developed specifically for video games that adapt agent behaviors in real-time. For example, the real-time NeuroEvolution of Augmenting Topologies (rtNEAT) method evolves artificial neural networks (ANNs) fast enough so that improvements can be perceived by the player. However, video game developers are accustomed to relying on prescripted behaviors, frequently encoded in finite state machines (FSMs). It is difficult to incorporate agents that develop behaviors on their own into the current practice. Such learned behaviors might be undesirable, violating the designer's intentions. This problem could be avoided if game designers could specify an initial behavior using an FSM and allow adaptation. This paper describes such a method, Knowledge-Based NEAT (KB-NEAT), which converts a FSM into an ANN using a KBANN-based technique. In this paper, KB-NEAT is tested in the game of blackjack, demonstrating that the FSM successfully converts into an ANN with identical behavior and further improves its performance during the game using NEAT. KB-NEAT can help the game industry utilize machine learning methods with minimal change to current practices.

1 Introduction

In many video games, non-player characters (NPCs) are driven by prescripted behaviors, often encoded as finite-state machines (FSMs) (Orkin 2002). Over the course of a game, players begin to recognize patterns these NPCs exhibit leading the game to become predictable and dull. Because they are controlled by scripts and FSMs, the NPCs have no way of adapting to these manipulations, and the game loses its appeal. In order to keep the game interesting and challenging, it is therefore important to develop methods that allow the NPCs to adapt in real-time.

Research on the NERO video game (NeuroEvolving Robotic Operatives) has achieved early success in achieving this goal (Stanley, Bryant, and Miikkulainen 2005). In NERO, NPCs adapt in real time to the player using the real-time NeuroEvolution of Augmenting Topologies (rtNEAT) method, which evolves increasingly complex artificial neural network (ANN) controllers. Thus, technology exists in NERO to allow real-time adaptation, but another issue has emerged. Adaptation from scratch can lead to a variety of behaviors, which is exactly what makes the game interesting, but these behaviors are no longer completely under the developer's control. There are no guarantees that a given level of performance is achieved and maintained, and unwanted side effects may emerge as well.

Instead of evolving the behaviors from scratch, it would therefore be useful to start the evolution from a set of scripted behaviors that establish the core performance, and let evolution improve them and generate variety based on them. This paper describes such a method, Knowledge Based NEAT (KB-NEAT), which combines the prescripted behaviors with the adaptability of a neural network using NEAT. The developers first describe the behavior in an FSM, and then convert it into an equivalent neural network using the Knowledge Based Artificial Neural Networks (KBANN) method (Towell and Shavlik 1994). This network serves as a basis for a population that is then further evolved with NEAT over the course of the game. In this way, developers can be confident that agents interact with the world based on the prescripted behaviors, but can also discover new behaviors if they turn out more effective during the actual game.

KB-NEAT is tested in this paper in the game of blackjack. One simple FSM and one complex FSM were hand-designed to play this game. These FSMs were converted into a neural network and allowed to adapt over a number of games. The results show that (1) KB-NEAT can improve the performance of the simple original behaviors, and (2) KB-NEAT does not hurt the performance if the original behavior is already well optimized. KB-NEAT therefore makes it possible to start a game with well-understood, prescripted behaviors and further improve them through evolution, allowing the game industry utilize machine learning with minimal change to current practices.

The next two sections explain NEAT and KB-NEAT. Section 4 describes the blackjack domain, and Section 5 overviews the experiments used to test KB-NEAT. The last sections cover the results of the experiments and discuss future opportunities.

2 NeuroEvolution of Augmenting Topologies (NEAT)

In order to allow agents to adapt in a video game, a powerful and reliable machine learning method is needed. Neuroevolution (NE), i.e. the artificial evolution of neural networks, is a promising such technology. In NE, a population of ANN controllers is evolved in a survival of the fittest competition. Because only the fittest ANNs are allowed to reproduce, behaviors gradually improve over generations. The method can discover successful solutions based on only high-level specifications of desired behavior, and through recurrent connections, complex behaviors that require memory can emerge.

NE has been successfully used to evolve motor-control skills such as those necessary in continuous-state games in many challenging non-Markovian domains (Floriano and Mondada 1994; Fogel 2001; Gomez and Miikkulainen 2003; Gruau et al. 1996; Moriarty and Miikkulainen 1996; Nolfi et al. 1994; Stanley and Miikkulainen 2004; Whitley et al. 1993). Recent NE research has focused on evolving higher-level strategies and including real-time adaptation, which are needed for both continuous and discrete multi-agent games (Agogino et al. 2000; Bryant and Miikkulainen 2003; Stanley and Miikkulainen 2004). One example is the NERO video game (Stanley, Bryant, and Miikkulainen 2005), where non-player characters evolve complex behaviors as the game is being played. NERO demonstrates that the technology can make commercial video games more interesting in the future.

The NERO game is based on the NEAT method of neuroevolution (Stanley and Miikkulainen 2002a; Stanley and Miikkulainen 2004), which is also used in the experiments in this paper. NEAT is well-suited for the task of improving upon prescripted behavior because it can evolve arbitrary network topologies. Methods such as KBANN that convert rules into a corresponding ANN topology can be used to construct a starting point for further evolution with NEAT.

NEAT is based on three key ideas. First, evolving network structure requires a flexible genetic encoding. Each genome in NEAT includes a list of connection genes, each of which refers to two node genes being connected. Each connection gene specifies the in-node, the out-node, the weight of the connection, whether or not the connection gene is expressed (an enable bit), and an innovation number, which allows finding corresponding genes during crossover. Mutation can change both connection weights and network structures. Connection weights mutate as in any NE system, with each connection either perturbed or not. Structural mutations, which allow complexity to increase, either add a new connection or a new node to the network. Through mutation, genomes of varying sizes are created, sometimes with completely different connections specified at the same positions.

Each unique gene in the population is assigned a unique innovation number, and the numbers are inherited during crossover. Innovation numbers allow NEAT to perform crossover without expensive topological analysis. Genomes of different organizations and sizes stay compatible throughout evolution, and the problem of matching different topologies (Radcliffe 1993) is essentially avoided.

Second, NEAT speciates the population, so that individuals compete primarily within their own niches instead of with the population at large. This way, topological innovations are protected and have time to optimize their structure before they have to compete with other niches in the population. The reproduction mechanism for NEAT is explicit fitness sharing (Goldberg and Richardson 1987), where organisms in the same species must share the fitness of their niche, preventing any one species from taking over the population.

Third, unlike other systems that evolve network topologies and weights (Gruau et al. 1996; Yao 1999) NEAT begins with a uniform population of simple networks with no hidden nodes. New structure is introduced incrementally as structural mutations occur, and only those structures survive that are found to be useful through fitness evaluations. This way, NEAT searches through a minimal number of weight dimensions and finds the appropriate complexity level for the problem.

In this paper, FSM blackjack players are converted into ANNs that NEAT further evolves to improve their performance. The next section describes the FSM to ANN conversion procedure.

3 KB-NEAT

The FSMs are converted to neural networks that are compatible with NEAT using a modified version of KBANN (Towell and Shavlik 1994; Maclin 1995). Instead of boolean inputs of the original KBANN (and its FSM version fsKBANN), the modified version processes continuously-valued inputs. Such inputs are necessary because video game states are often described with continuous variables, and also because the FSM conditionals can be represented more accurately.

Each FSM contains a list of inputs, outputs, conditionals and states (Carlisle 2002). Each state represents a behavioral action to be performed by agents in the game. Conditionals are used to determine transitions from one state to another. KB-NEAT currently supports 5 different conditional types: (1) less-than, (2) less-than-equal-to, (3) greater-than, (4) greater-than-equal-to, and (5) AND. These conditionals receive input variable values and constant floating-point values as their input and evaluate to true or false. Inputs are normalized floating-point values between a minimum and a maximum that describe the state of the game.

The FSM can be expressed as a neural network structure: The output nodes of the ANN correspond to the

states of the FSM, and the input nodes correspond to all the possible variables used in the FSM as well as the previous values of the output nodes. An FSM can be converted into an equivalent ANN in five steps (Figure 1 depicts an example of this process). Steps 1 through 4 convert the FSM to a set of rules and instantiate the proper inputs and outputs so that the ANN can implement the same state transitions as the FSM; step 5 translates the rules into their corresponding ANN structure.

1. Extract a list of rules from the FSM by traversing over every transition. At each transition, a conditional determines whether the transition should be taken or not; this conditional becomes the left-hand side of the rule. Panel 1 in Figure 1 shows the list of rules generated from the complex FSM.

2. Create a network with the appropriate number of inputs and outputs. The number of output nodes is equivalent to the number of states in the FSM. The number of input nodes is equal to the sum of the number of inputs available, the number of states in the FSM, and one for the bias. Assign every node in the network with a number designating its node id.

3. Take the list of inputs and states and match them to the corresponding nodes in the network created in step 2. In panel 3, the states used as inputs are marked with an apostrophe, indicating that these inputs represent the previous value of the output node with the same name.

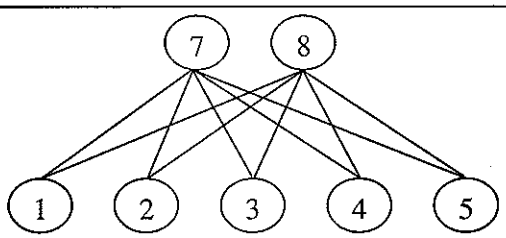
4. Go through the rules generated in step 1 and replace the names of all inputs with the appropriate node using the table generated in step 3. This step makes it easier to modify the network later.

5. Add the rules to the neural network one at a time. In this step, each of the five conditionals is converted into an appropriate ANN structure specific to that conditional.

Step 5 is a modified version of KBANN. Recall that each of the five conditionals is converted into a different

{ if HANDVALUE <= .5333 and HANDVALUE >= .4 and DEALERSUP <= .5454 and HIT' > 0.5 then {output STAND 1.0 } }
{ if HANDVALUE >= .5666 and HIT' > 0.5 then {output STAND 1.0 } }

Panel 1: Conditionals are converted into rules. The general rule is:
{ if CONDITIONAL and CURRENT_STATE' > 0.5 } then {output NEXT_STATE 1.0}



Panel 2: Construct a NN. The number of output nodes is the number of states. The number of input nodes is sum of the number of inputs, states and one for the bias.

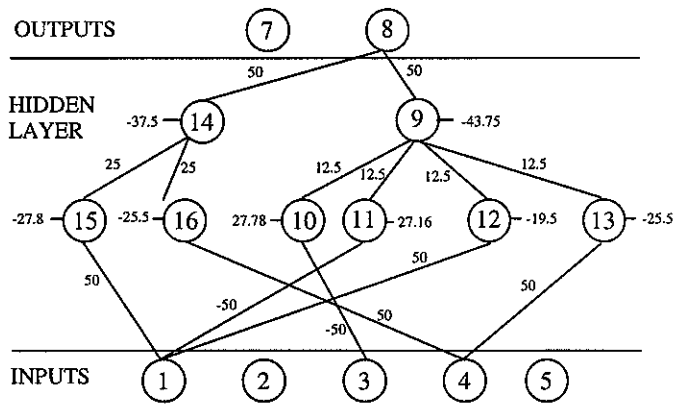
INPUTS
1 – HANDVALUE
2 – HASACES
3 – DEALERSUP
4 – HIT'
5 – STAND'
6 – BIAS

OUTPUTS
7 – HIT
8 – STAND

Panel 3: Gather a list of all the inputs and the states and assign them to the appropriate node ids.

{ if 1 <= .5333 and
1 >= .4 and
3 <= .5454 and
4 > 0.5 then {output 8 1.0 } }
{ if 1 >= .5666 and
4 > 0.5 then {output 8 1.0 } }

Panel 4: The input and state names from the set of rules in panel 1 are replaced with the corresponding node IDs.



Panel 5: Converting the rules to the NN. Connections with 0 weight not shown.

Figure 1. The KB-NEAT algorithm demonstrated using the complex FSM from Figure 2b.

network substructure. While in KBANN the only conditionals are boolean operators such as AND, KB-NEAT also supports conditionals that compare floating point numbers. In steps 1 through 4, the FSM was converted into a set of conditionals, all of which must evaluate to true if the specified state transition is to occur (Figure 1, panel 1 shows the general form of a state transition rule, and Panel 4 gives a specific example.). For each conditional in the set, a hidden node is created in the network. The input for the hidden node includes one link to a node representing the variable and a bias representing the constant used for comparison.

For a *greater-than* or *greater-than-equal-to* comparison, the link's weight is positive 50 and the bias is set to $-50c-\varepsilon$, where c is the constant in the conditional and ε is a small offset. *Less-than* and *less-than-equal-to* are similarly converted except with a negative weight and a positive bias.

The second layer of hidden nodes represent conjunctions (i.e. using AND) of the conditionals in each rule. For conjunctive rules, the weights are $50/n$, where n is the number of inputs to the hidden node. The bias is then set to $25/n - 50$. For disjunctive rules, the weights are assigned $50/n$ as well, but the bias is always set to $-25/n$. Following this procedure, the ANN perfectly replicates the policy of the FSM.

Finally, a method is needed to implement FSM state transitions using the ANN. In a FSM, if any of the transitions from the current state are activated a transition to another state occurs. To determine the next state of the ANN, the output node with the highest activation is chosen as the next state if the activation value exceeds a threshold (0.6 in this paper). If no output nodes are above the threshold, the ANN does not transition and hence remains in the same state.

The ANN created from the FSM can be improved through further evolution in NEAT. A population of such networks is created by perturbing these networks slightly, and this population will serve as a starting point for evolution. The test domain of blackjack is described next.

4 The Blackjack Domain

Blackjack was chosen as the domain for testing KB-NEAT because it is easy to design FSMs with different skill levels to play the game.

Blackjack is a simple card game where players try to obtain a hand value as close as possible to 21 but still below it, while beating the dealer's hand value. In the simulation, a single deck is shuffled before every game. Each agent is dealt two cards and the dealer is dealt two cards, but the agent can only see the value of one of those cards. The agent must decide whether to hit or stand based upon the cards they have and the card the dealer has facing up. If the agent chooses to hit, a card is dealt to it and the decision process starts over again. If the cards total over 21 (with face cards being valued at 10 and an ace counting as 11 or 1), the agent has busted and loses the hand. After all the agents have either busted or stood, the dealer hits until it busts or has a hand totaling 17 or higher.

In the simulation, the dealer is controlled by the simple FSM shown in Figure 2a. This same strategy, called the simple FSM, was also used as one of the prescripted behaviors. The other prescripted behavior is shown in Figure 2b. This FSM, called the complex FSM, represents a previously known effective strategy (Smith 2004). Since it is known that the complex FSM can perform better than the simple FSM, an important question is whether evolution can improve the simple FSM to a similar level of play. Second, if evolution starts with the complex FSM, it should maintain a similar level of play and not get

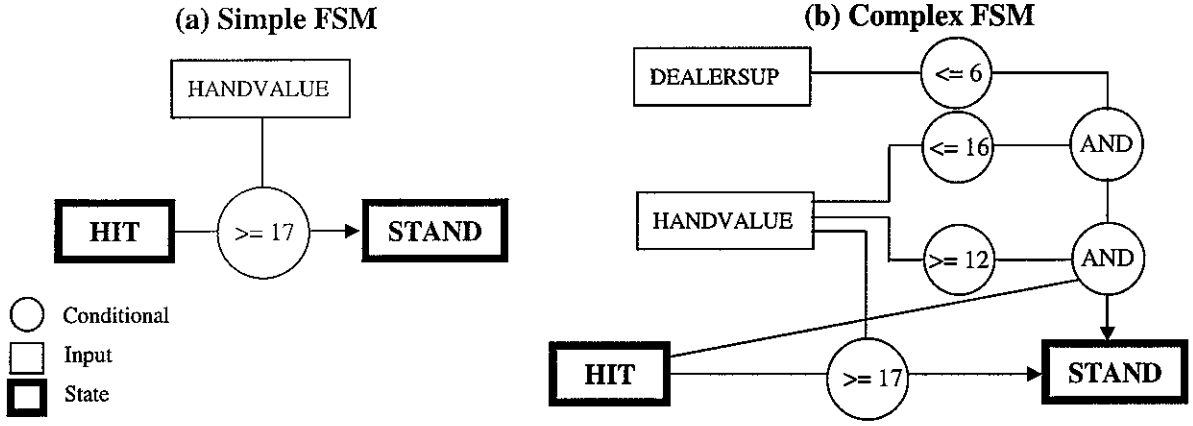


Figure 2. (a) The simple FSM has one conditional (>= 17) that takes the player's hand value as an input. This FSM will stand when 17 is reached. (b) The Complex FSM uses 5 conditionals (in one case joining 3 different conditionals together using two ANDs) to describe a more successful way of playing blackjack.

weaker over time. The experiments were designed to answer these two questions.

5 Experiments

Three experiments were performed. The first one was designed to verify that the conversion from FSM to an ANN is indeed correct, i.e. that the ANN duplicates the behavior of the FSM. The second one answers the question: Is KB-NEAT able to improve upon the performance described in a simple FSM? The third experiment in turn addresses the question: If the FSM solution is nearly optimal, does further NEAT evolution hurt performance?

In the first experiment, a single ANN formed with KB-NEAT was placed into a game against the FSM from which it was derived: The simple FSM in the first part of the experiment, and the complex FSM in the second. The two agents then played one million games. The ANN and FSM were dealt exactly the same cards in each hand, causing them to make decisions based on the same circumstances. The number of hands each agent won and the way each hand was played was compared over the simulation.

In the second experiment, a population of ANNs was formed from an ANN derived from the simple FSM, and in the third, from the complex FSM. In both experiments, an initial population of 100 networks consisted of the converted network and 99 versions of it with weights uniformly perturbed with a mutation parameter of 0.1. Because the converted network remained in the population, the best network in the population cannot be worse than the original FSM. Each network was evaluated on its performance over 5000 games of blackjack. Fitness was determined as the square of the percentage of games won. The best networks at the end of evolution were then tested in one million games, and their performance compared to that of the originals FSM. The performance was averaged over 20 runs.

6 Results

In the first experiment, the NN and the FSM played every hand exactly the same way with both conversions (simple and complex). Over 1 million games both agents had the same number of wins and played each hand the same.

The second experiment, shown in figure 3, demonstrated that it is possible for NEAT to evolve networks that improve upon the original converted ANN. Using KB-NEAT, the ANNs derived from the simple FSM improved 2 percentage points over 500 generations. The difference is significant well above the 95% confidence level. In fact, the evolved ANNs were very close to the performance of the complex FSM.

The ANNs derived from the complex FSM maintained a high level of play throughout the run, declining only slightly in their performance (just below the 95% confidence range). The difference is due to noisy fitness estimation. With only 5000 hands, occasionally a network appears to be playing better than it actually is, and is selected as a champion. Had the number of hands been increased from 5000, the performance would have been closer to the original FSM, and possibly surpassed it. In fact, in five of the 20 cases, performance remained the same, and in four other cases, evolution improved slightly upon the performance of the complex FSM. Of course, more accurate evaluations take more time, so there is a tradeoff on how quickly the adaptation has to happen and how accurate it needs to be.

In sum, the performance improved significantly over

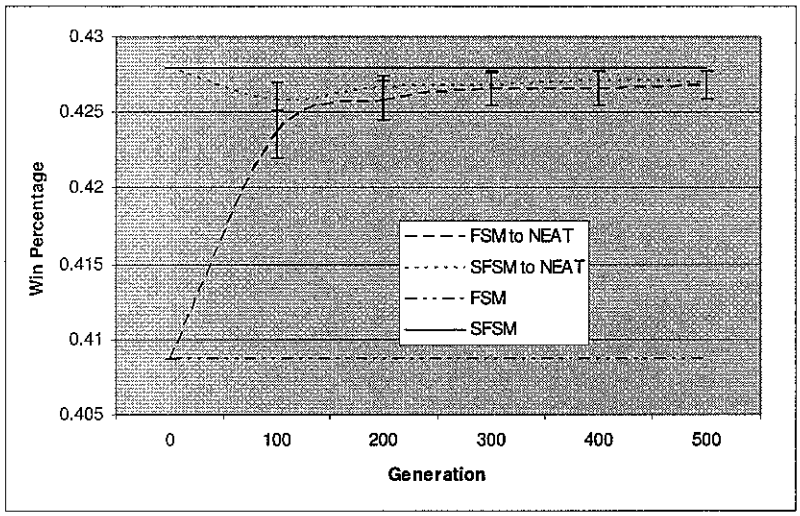


Figure 3. This graph shows winning percentage improvements over each generation. The FSM and SFSM lines are both flat, as they do not evolve behaviors. The FSM to NEAT line rises dramatically above the FSM it was based upon, well above the 95% confidence bars. However, the SFSM to NEAT line dips below the SFSM line just slightly. The corresponding network is shown in figure 1.

the simple FSM and remained strong even when evolving from the complex FSM, demonstrating that improving on prescripted behaviors using KBANN is both possible and safe.

7 Current Work

The results of the blackjack experiment demonstrate a potential for KB-NEAT to be used in more complex real-time simulations. The NERO project has created a framework to easily integrate and test KB-NEAT in a robust video game environment. Currently the entities controlled in NERO are trained off a population containing ANNs with randomly perturbed weights. By integrating KB-NEAT into NERO, training should become more focused as entities will be able to begin with a basic knowledge of the world around them. Players will not have to wait for the population to learn basic maneuvers such as approaching an enemy or shooting, before carrying on to more complex tasks such as obstacle navigation or fighting tactics.

Following a successful integration into NERO, experiments will be set up to further test the abilities of KB-NEAT. The first experiment to be conducted will use evolution to develop a team of enemies pitted against a single opponent. The details of the experiment are as follows. A large number of enemies will spawn in random locations around a simple environment. In addition to these enemies, a powerful single agent, hereafter known as the hero, will be spawned in a randomly location. The enemies and the hero will be controlled by an ANN derived from a FSM using KB-NEAT. The agents are set loose and once either the hero kills all the enemies, the hero is killed or a time limit is exceeded the simulation ends. Following the termination of the simulation, the enemy brains are evaluated and a new population is created for the next round. The hero brain is evaluated and another brain extracted from the hero population. Over the course of many games the hero and the enemies will co-evolve to better fight each other.

With the usage of KB-NEAT, a FSM must be created to describe the actions of the first enemies and hero. The complexity of the initial FSMs will be much greater than that of the FSM machines used to describe Blackjack Strategy. At the moment the states an agent may exhibit are move forward, move backward, turn left, turn right and fire. Each agent also has access to more than twenty sensors comprising of data relating to enemy proximity, friend proximity, and obstacle obstruction. From initial estimates the FSMs for these agents will consist of more than fifteen transitions between the five states. This is much more complex than two machines describing blackjack strategy which consist of a combined seven conditionals and three state transitions.

This richer domain closely resembles current video game environments unlike the Blackjack simulation presented in this paper. This type of system where enemies are encountered in groups at a time is present in many current video games. By conducting experiments in this domain, the results will be more convincing and appealing to video game developers.

8 Discussion and Future Work

KB-NEAT allows game developers to easily convert their FSM-driven agents to adaptable agents controlled by ANNs. The method therefore allows both to control what kind of behavior is likely to emerge and to allow better behaviors to evolve. In games with a lot of human interaction with NPCs, such as a Massive Multiplayer Online Role Playing Game (MMORPG), the ability for NPCs to slowly adapt to humans would keep the game interesting and challenging over the course of months and even years. Moreover, the ability to start from a prescripted behavior means that games will function from the beginning as the developer intended.

One immediate direction of future work is to try to counteract the effect of noisy evaluations by continually inserting the original converted FSM into the population as the game progresses. That way, the original performance level would be guaranteed to remain in the population throughout the run.

The method could also be extended so that it can learn new behavioral states and inputs that were originally not included in the FSM. For example, the blackjack domain could be further expanded to allow for more complex playing styles. Adding the ability to split, double down, and get insurance would greatly expand the ways to improve upon the FSM strategy. With an expanded game, even without knowledge of splitting or doubling down in the FSM, the knowledge-based ANN should be able to adapt to use these new actions to its advantage. How to best balance the initial prescriptions and future learning ability is an interesting direction for future work.

9 Conclusion

KB-NEAT allows game developers to use machine learning to control NPCs with minimal change to current practice. By converting prescripted behaviors into equivalent ANNs, developers maintain control of NPC behavior. During the course of the game, the agents can further adapt to the world, while still being controlled by the high-level behavioral structures designed by the developers. Using KB-NEAT, it is possible to significantly improve performance over the initial FSM, and even with a highly optimized initial state, the performance remains strong. KB-NEAT can therefore serve as a vehicle for bringing sophisticated machine learning into the practice of game development.

10 Acknowledgements

This research was supported in part by DARPA under NRL Grant N00173041G025 and by the Digital Media

Collaboratory. Special thanks for Chern Han Yong, who developed the code to add rules to NEAT networks. Also, thanks for Kenneth Stanley for developing the rtNEAT algorithm and helping me through every step of this project.

Reference

Agogino, A., Stanley, K., and Miikkulainen, R. 2000. Real-time interactive neuro-evolution. *Neural Processing Letters*, 11:29–38.

Bryant, B. D., and Miikkulainen, R. 2003. Neuroevolution for adaptive teams. In *Proceeedings of the 2003 Congress on Evolutionary Computation (CEC 2003)*, vol. 3, 2194–2201. Piscataway, NJ: IEEE.

Carlisle, P., and Rabin S. ed. 2002. Designing a GUI Tool to Aid in the Development of Finite-State Machines. *AI Game Programming Wisdom.* Hingham, Mass.: Charles River Media, INC..

Floriano, D., and Mondada, F. 1994. Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot. In *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior.*

Fogel, D. B. 2001. *Blondie24: Playing at the Edge of AI.* San Francisco, CA: Morgan Kaufmann.

Goldberg, D.E., and Richardson, J. 1987. Genetic algorithms with sharing for multimodal function optimization. In Grefenstette, J.J., ed: *Proceedings of the 2nd Intl. Conf. on Genetic Algorithms*, San Francisco, CA: Morgan Kaufmann 148-154.

Gomez, F., and Miikkulainen, R. 2003. Active guidance for a finless rocket using neuroevolution. In *Proceeedings of the Genetic and Evolutionary Computation Conference (GECCO-2003).* San Francisco, CA: Morgan Kaufmann.

Gruau, F., Whitley, and D., Pyeatt, L. 1996. A comparison between cellular encoding and direct encoding for genetic neural networks. In Koza, J.R., Goldberg, D.E., Fogel, D.B., Riolo, R.L., eds.: *Genetic Programming 1996: Proceedings of the First Annual Conference*, Cambridge, MA, MIT Press 81-89.

Maclin, R. 1995. Learning from Instruction and Experience: Methods for Incorporating Procedural Domain Theories into Knowledge-Based Neural Networks. PhD thesis, Department of Computer Sciences, University of Wisconsin-Madison.

Moriarty, D. E., and Miikkulainen, R. 1996. Evolving obstacle avoidance behavior in a robot arm. In Maes, P., Mataric, M. J., Meyer, J.-A., Pollack, J., and Wilson, S.W., editors, *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, 468–475. Cambridge, MA: MIT Press.

Nolfi, S., Elman, J. L., and Parisi, D. 1994. Learning and evolution in neural networks. *Adaptive Behavior*, 2:5–28.

Orkin, J., and Rabin, S. ed. 2002. 12 Tips from the Trenches. *AI Game Programming Wisdom.* Hingham, Mass.: Charles River Media, INC..

Radcliffe, N. J. 1993. Genetic set recombination and its application to neural network topology optimization. *Neural computing and applications*, 1(1):67–90.

Smith, K. R., 2004. website: Blackjack Basic Strategy Chart. http://www.blackjackinfo.com/cgi-bin/bjbse.cgi. January 2005.

Stanley, K.O., and Miikkulainen, R. 2002a. Evolving neural networks through augmenting topologies. *Evolutionary Computation* 10:99-127.

Stanley, K.O., and Miikkulainen, R. 2002b. Efficent reinforcement learning through evolving neural network topologies. In *Proceedings of the Genetic and Evolutionary Computation Conf. (GECCO-2002)*, San Francisco, CA, Morgan Kaufmann.

Stanley, K.O., and Miikkulainen, R. 2004. Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research*, 21:63-100.

Stanley, K.O., Bryant, B. D., and Miikkulainen, R. 2005. The NERO Real-Time Video Game. *IEEE Transactions on Evolutionary Computation Special Issue on Evolutionary Computation and Games.*

Towell, G. G., and Shavlik, J. W. 1994. Knowledge based artificial neural networks. *Artificial Intelligence* 70:119-165.

Whitley, D., Dominic, S., Das, R., and Anderson, C. W. 1993. Genetic reinforcement learning for neurocontrol problems. *Machine Learning*, 13:259–284.

Yao, X. 1999. Evolving artificial neural networks. *Proceedings of the IEEE* 87(9): 1423-1447.